05

Interfaces repository

Transcripción

[00:00] Ya explicamos que lo que necesitamos ahora es crear la integración, la capa que nos va a permitir pasar de esto a registros en la base de datos usando esto. Entonces les expliqué que antiguamente teníamos el patrón DAO en el cual tenías que crear tu entity manager, definir tu conexión con base de datos, crear la conexión, cerrar la conexión y era un trabajo muy engorroso.

[00:28] Ahora lo que vamos a hacer es crear repositorios. ¿Cómo es eso? Entramos aquí al médico y le voy a dar clic derecho y voy a crear una nueva interfaz en este caso: MedicoRepository, vamos a crear repositories y el del tipo interface. Entonces con esta interfaz nosotros vamos a ser capaces de hacer todo el proceso de gestión con la base de datos a nivel del CRUD: crear, guardar objetos, listar, actualizar, etcétera, pero automáticamente. ¿Por qué?

[01:09] Porque vamos a extender de otra interfaz llamada JpaRepository y está interfaz es propia de Spring data también, pero JpaRepository necesita dos parámetros. El primer parámetro es el tipo de objeto que yo voy a guardar aquí, por ejemplo, en este caso es médico, que es la entidad que yo voy a salvar, es el tipo de entidad con el que yo voy a trabajar en este repositorio.

[01:36] Segundo, necesito el tipo de objeto del id. Entonces en este caso sería un Long. Entonces, si se dan cuenta si reconocen lo que es esto, estos son generics. Eso quiere decir que si aquí entró a JpaRepository, primero que todo vemos que usa los generics aquí, T, tipo de objeto, id, el id que voy a usar y tiene métodos aquí como saveAndFlush, usando los generics, delete, deleteAll, getOne, getById en caso necesite uno, está deprecado.

[02:14] Hay otros métodos más eficientes, pero vemos que findAll, usando los mismos generics. Vemos que está interfaz nos brinda métodos que van a ser muy útiles en el futuro y yo no voy a necesitar escribir ni una sola línea de código. Así de simple, ni una sola línea de código. Yo tengo mi repositorio. Eso es todo lo que tengo que hacer. No tengo que hacer nada más.

[02:43] Ahora lo que voy a hacer es entrar a MedicoController y llamar a ese repositorio, ¿cómo lo hago? Simple. Voy a crear a mi MedicoRepository, voy a llamarlo aquí y Spring tiene una anotación especial para decir Spring, necesito que de tu contexto busques este objeto de aquí, este MedicoRepository y me lo envíes, me lo inyectes, para no tener que crear uno nuevo.

[03:15] Si no, yo tendría que darle digamos un igual new y la implementación del MedicoRepository, la cual no la tengo. Spring la tiene internamente. Para eso yo voy usar una anotación autowired. Vale resaltar aquí que no es recomendable por fines de testing, no es recomendable usar autowired a nivel de la declaración del parámetro del atributo.

[03:41] No es recomendable porque si ustedes quieren usar pruebas unitarias con esto, van a tener problemas, va a ser muy difícil que puedan hacer un mock de MedicoRepository. En fin, este no es un curso de testing pero es un consejo que les doy. Ahora por fines didácticos no lo vamos a hacer. Otra forma de hacer la inyección de dependencias con autowired sería con setter, pero lo vamos a dejar así por ahora.

[04:08] Pero recuerden: esta forma es una forma abreviada para fines didácticos, pero no se las recomiendo usarla en la práctica. Volvamos al tema. ¿Y aquí qué es lo que tengo que hacer? Simple. Tengo mi MedicoRepository y aquí cuando tengo acá le voy a dar un save. ¿Save de qué? Save de datosRegistroMedico y vemos que me está dando un error de compilación. ¿Por qué?

[04:40] Porque aquí si se dan cuenta save usa el generic que yo le estoy enviando como parámetro. Si vengo a mi MedicoRepository, como parámetro yo le estoy mandando un objeto médico y no un objeto DatosRegistroMedico, entonces ahí yo tengo ya un digamos un conflicto. Yo tengo dos opciones aquí.

[05:04] La primera crear un método que me traduzca y me haga el mapeo, digamos así, de DatosRegistroMedico a médico o la otra solución, simplemente voy a cargarle un constructor, voy a entrar aquí, voy a crearle un constructor a médico, new Medico que acepte como parámetro los datos de registro del médico. Aquí podría ser tranquilamente, datosRegistroMedico.getNombre y etcétera.

[05:39] Pero no es lo que yo quiero hacer por ahora porque si no mi método quedaría muy extenso. Entonces lo que voy a hacer directamente es decirle: "médico constrúyete un constructor que acepte como parámetro un dato registro médico". Y es lo que voy a hacer en este momento.

[05:59] Vengo a Medico. Y le voy a decir create constructor y aquí vemos, pues que ya tiene un public Medico(DatosRegistroMédico). Aquí lo que voy a hacer es hacer el mapeo de this.nombre = datosRegistroMedico.nombre(), this.email = datosRegistroMedico.email(), this.documento = datosRegistroMedico.documento(), this.especialidad, la misma historia .especialidad, porque ese mismo tipo de objeto entonces no hay ningún problema.

[06:54] Y finalmente this.dirección = datosRegistroMedico.direccion. Y vemos que aquí me está dando un error de compilación, porque él necesita un objeto de tipo dirección pero yo le estoy dando datos dirección. Entonces aquí yo tengo otro pequeño issue con esto.

[07:17] Para eso yo lo que voy a hacer aquí es darle un new Dirección y esa dirección va a recibir como parámetro los datos de registro de Dirección. No lo copié bien, le voy a dar cortar, voy a darle aquí "Ctrl + X". Listo. Y la misma

historia. ¿Qué voy a hacer aquí? Simple, voy a darle un constructor a dirección y repetimos exactamente lo mismo aquí.

[07:48] this.calle = direccion.calle(), this.numero = dirección.número(), this.distrito = dirección.distrito(), this.complemento=dirección.complemento(). Vemos que tenemos un error de compilación de número y debe ser por el tipo de dato. Y finalmente tenemos ciudad. Eso lo vamos a copiar así, para no tener que estar escribiendo todo a cada momento.

[08:40] Vemos que en número me está dando un error de compilación porque del DTO que yo estoy enviando, acá String, para no complicarme la vida voy a usar solo strings aquí. No va a tener ningún impacto. Ya tengo mi médico bien mapeado. Tengo mi dirección mapeada y tengo mi repositorio creado.

[09:05] Parece que todo está bien implementado. Ahora voy a venir a mi controller. Mi controller también ya no tiene ningún tipo de error, ya compila todo perfectamente, tengo mi MédicoRepository, tengo mi Medico entidad que está recibiendo el DTO. Vamos a probar este código, vamos a ejecutarlo. Entonces le doy a play. Vamos a play, vemos que está iniciando.

[09:40] Y la aplicación inició en 4 segundos. Vamos a dar limpiar esto, no necesitamos, venimos a Insomnia y disparamos nuestro request para ver si lo va a guardar en la base datos. Y sale un 500 internal server error. Alguna cosa sucedió en el back end y vamos a ver aquí qué es lo que nos está diciendo.

[10:01] Dice la tabla 'vollmed_api.medicos', o sea la tabla que hemos definido aquí en nuestro entity, no existe. ¿Por qué? Porque si vemos aquí, en nuestro UI de la base de datos, en nuestro cliente de base de datos, tenemos la base de datos vollmed_api. Perfecto, existe, pero no nos creó ninguna tabla. No hay ninguna tabla. Entonces aquí tenemos nuevamente igual dos opciones.

[10:35] Podemos tranquilamente usar el cliente visual, darle a new table y crear la tabla, ejecutar el código o podemos usar una herramienta que pueda hacer

eso por nosotros y tenerlo versionado. Alerta de spoiler. Es Flyway. Y eso lo vamos a ver en el siguiente video.